UDC 004.9: 004.05
DOI: 10.31891/CSIT-2021-5-14

I. LOPATTO, M. LEBIGA, T. HOVORUSHCHENKO
Khmelnytskyi National University

METHOD OF ESTIMATING THE LABORIOUSNESS OF THE PROCESS OF DEVELOPING COMPUTER SYSTEMS' SOFTWARE

The paper proposes a method for estimating the laboriousness of software development based on functional points, which allows to determine the number of functional points for a software project, and also allows in the early stages of the life cycle to estimate the size of a software project (for example, LOC-assessment). The developed method eliminates the dependence of evaluation on the subjects involved in the evaluation process.

Keywords: software, software project, laboriousness of software development, LOC-estimation, functional point, method of functional points.

І. ЛОПАТТО, М. ЛЕБІГА, Т. ГОВОРУЩЕНКО
Хмельницький національний університет

МЕТОД ОЦІНЮВАННЯ ТРУДОМІСТКОСТІ ПРОЦЕСУ РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ

Оцінювання трудомісткості процесу розроблення програмного забезпечення (ПЗ) комп'ютерних систем є одним з найбільш важливих видів діяльності в процесі створення ПЗ. За відсутності адекватної та достовірної оцінки неможливо забезпечити чітке планування та управління проектом. Недооцінка вартості, тривалості та ресурсів, необхідних для створення ПЗ, тягне за собою недостатню чисельність проектної команди, надмірно стислі терміни розроблення і, як результат, втрату довіри до розробників у випадку порушення графіку.

У статті запропоновано метод оцінювання трудомісткості розроблення програмного забезпечення на основі функційних точок, який дає можливість визначити кількість функційних точок для програмного проекту, а також дозволяє на ранніх етапах життєвого циклу оцінити розмір програмного проекту (LOC-оцінка) при його потенційній реалізації різними мовами програмування. Розроблений метод усуває залежність оцінки від суб'єктів, залучених до процесу оцінювання.

Розроблений метод оцінювання трудомісткості на основі функційних точок відрізняється від існуючих методів тим, що є формалізованим, за рахунок чого усуває залежність оцінки від суб'єктів, залучених до процесу оцінювання. Переваги розробленого методу: можливість його використання на ранніх етапах життєвого циклу ПЗ; відсутність залежності від використовуваної мови програмування, методології та технології; можливість вимірювання LOC-оцінок для реалізації проекту різними мовами програмування; можливість застосування даного методу для програмних систем із графічним інтерфейсом користувача; врахування факторів середовища за потреби.

Перспективою для подальших досліджень є: розроблення методу (способу, алгоритму) оцінювання факторів середовища; розроблення методу оцінювання трудомісткості розроблення ПЗ на основі точок властивостей; розроблення методу (способу, алгоритму) оцінювання тривалості та вартості розроблення ПЗ на основі отриманих розробленими методами LOC-оцінок; розроблення системи підтримки прийняття рішень для оцінювання трудомісткості процесу розроблення ПЗ комп'ютерних систем на основі функційних точок та точок властивостей, в основу якої будуть покладені розроблені математичні методи.

Ключові слова: програмне забезпечення (ПЗ), програмний проект, трудомісткість розроблення ПЗ, LOC-оцінка, функційна точка, метод функційних точок.

Introduction

Estimating the laboriousness of software development is one of the most important activities in the software development process. In the absence of an adequate and reliable assessment, it is impossible to ensure clear project planning and management. Underestimation of the cost, duration and resources required to create software entails insufficient number of project team, excessively short development time and, as a result, loss of confidence in developers in case of schedule violations [1-4].

Models and methods of the laboriousness assessment are used to solve many problems, among which the following can be distinguished [5]: development of the project budget (with required accuracy); analysis of the degree of risk and the choice of a compromise solution (with specifying the scope of the project, the possibility of reuse, the number of developers, the equipment used); project planning and management (the obtained results provide the distribution and classification of costs by components, stages and operations); cost analysis to improve the quality of software (allows you to estimate the costs and benefits of the investment strategy in improving technology and reusability).

The procedure for estimating the laboriousness of software development consists of the following steps [5]: estimating the size of the developed product; assessment of the laboriousness (in man-months or man-hours); estimation of project duration (in calendar months); project cost estimation.

When assessing the laboriousness of software development, there are currently the following problems [5]:

- dependence of evaluation on the subjects involved in the evaluation process;
- lack of a database of evaluations of standard projects (especially for fundamentally new software projects) and decision support systems (DSS) to assess the laboriousness of software development (although the accumulation

of a large number of evaluations for projects of different types would greatly simplify the task of evaluating laboriousness);

- the use of LOC-estimates (for ready-made software code) as the most common unit of measurement of software size with a large number of disadvantages of such assessment: the inability to use in the early stages of the software life cycle, when there is no ready-made program code; dependence of LOC-assessment on the development environment and design methods; not including costs not related to the program code; non-considering the amount of code automatically generated and generated by the programmer; when using LOC-estimates as the main unit of measurement, the laboriousnessy of software development is calculated only for ready-made program code;
- single use of estimating the laboriousness of software development based on functional points (mainly due to the lack of mathematical apparatus for such an evaluation process), but functional points as the main unit of measurement allow calculating the laboriousness at the early life cycle stages and predict LOC-estimates of software for implementation in different languages, i.e. support a reasonable choice of programming language for implementation.

Given the above, *the urgent task* now is the development of the method for support the process of assessing the laboriousness of software based on functional points, through which functional points can become the main unit of assessment of laboriousness, which will assess the laboriousness of software development and predict its LOC-assessments in the early stages of lifecycle, when there is no ready-made program code, as well as to eliminate the dependence of evaluation on the subjects involved in the evaluation process.

Then *the purpose of this study* is the development of the method of estimating the laboriousness of the process of developing computer systems' software based on functional points.

Method of estimating the laboriousness of the process of developing computer systems' software

The size of the software is best estimated in terms of the number and complexity of functions implemented by program code, rather than by the number of lines of code [5]. Functional point is a unit of measurement of software functionality, which is a combination of software properties such as: the intensity of use of input and output of external data, system interaction with the user, external interfaces and files used by software [5]. When using function points, the categories of user business functions are subject to measurement. In [5] the method of estimating the laboriousness of software development based on functional points in the form of recommendations is given, the necessary categories and weights are also given, but there is no formalized method that leads to free use and interpretation of this method, so the estimates of the laboriousness depend on the actors involved in the evaluation process..

The method of estimating the laboriousness of the process of computer systemsэ software development based on functional points consists of the following stages [5]:

1) determining the number of functions by category;
2) determining the number of data elements by category;
3) determining the number of involved files by category;
4) determining the number of logical records of the type "format-relationship" by category;
5) determining the number of functions for each category by levels of complexity
6) check the number of functions for each category;
7) determining the number of functional points for the functions of each category;
8) determination of the total number of functional points;
9) determining the user's need to correct the total number of function points;
10) assessment of environmental factors;
11) determination of the general assessment of factors of the environmental ;
12) determination of the correction factor of complexity;
13) correcting the number of functional points;
14) assessment of the LOC indicator for a software project with its potential implementation in different programming languages.

Let's detail and formalize the stages of the method of estimating the laboriousness of software development based on functional points:

1) determining the number of functions by category:

qt_{fi} - number of input functions (input functions from the end user);

qt_{fo} - number of output functions (output functions intended for the end user);

qt_{cfi} - the number of input requests (input requests are external specific commands or requests that are executed by software and generated from the outside; it is the requests that provide direct access to the database);

qt_{cfo} - the number of output requests (output requests are external specific commands or requests that are executed by software and generated externally; it is the requests that provide direct access to the database);

qt_{f} - the number of files (data structures that are the primary logical group of user data, which are constantly completely inside the software system and available to users through input, output, queries or interfaces);

qt_i - number of interfaces (all data streams, including those stored outside the software system)..

For the possibility of application in the early stages of the life cycle, the specified number of functions by category will be determined on the basis of the specification of software requirements;

2) determining the number of data elements by categories: $QTED_{fi} = \{ qted_{fi_1},...,qted_{fi_{qt_{fi}}} \}$ - the set of quantities of data elements for input functions; $QTED_{fo} = \{ qted_{fo_1},...,qted_{fo_{qt_{fo}}} \}$ - the set of quantities of data elements for output functions; $QTED_{cfi} = \{ qted_{cfi_1},...,qted_{cfi_{qt_{cfi}}} \}$ - the set of data elements for input requests; $QTED_{cfo} = \{ qted_{cfo_1},...,qted_{cfo_{qt_{cfo}}} \}$ - the set of data elements for output requests; $QTED_f = \{ qted_{f_1},...,qted_{f_{qt_f}} \}$ - the set of quantities of data elements for files; $QTED_i = \{ qted_{i1},...,qted_{i_{qt_i}} \}$ - the set of quantities of data elements for interfaces;

3) determining the number of involved files by categories: $QTF_{fi} = \{ qtf_{fi_1},...,qtf_{fi_{qt_{fi}}} \}$ - the set of numbers of involved files for input functions; $QTF_{fo} = \{ qtf_{fo_1},...,qtf_{fo_{qt_{fo}}} \}$ - the set of quantities of involved files for output functions; $QTF_{cfi} = \{ qtf_{cfi_1},...,qtf_{cfi_{qt_{cfi}}} \}$ - the number of involved files for input requests; $QTF_{cfo} = \{ qtf_{cfo_1},...,qtf_{cfo_{qt_{cfo}}} \}$ - the set of numbers of files involved for output requests;

4) determining the number of logical records of the type "format-relationship" by categories: $QTLR_{fl} = \{ qtlr_{f_1},...,qtlr_{f_{qt_f}} \}$ - the set of numbers of logical records for files; $QTLR_i = \{ qtlr_{i_1},...,qtlr_{i_{qt_i}} \}$ - the set of numbers of logical records for interfaces;

5) determining the number of functions for each category by levels of complexity (there are three levels of complexity: s - simple, m - medium, c - complex): qt_{fi_s} - the number of simple input functions; qt_{fi_m} - the number of input functions of medium difficulty; qt_{fi_c} - the number of complex input functions; qt_{fo_s} - the number of simple output functions; qt_{fo_m} - the number of output functions of medium difficulty; qt_{fo_c} - the number of complex output functions; qt_{cfi_s} - the number of simple input requests; qt_{cfi_m} - the number of input requests with a medium level of complexity; qt_{cfi_c} - the number of complex input requests; qt_{cfo_s} - the number of simple output requests; qt_{cfo_m} - the number of output requests with the medium level of complexity; qt_{cfo_c} - the number of complex output requests; qt_{f_s} - the number of simple files; qt_{f_m} - the number of files of medium complexity; qt_{f_c} - the number of complex files; qt_{i_s} - the number of simple interfaces; qt_{i_m} - the number of interfaces of medium complexity; qt_{i_c} - the number of complex interfaces.

In all the above variables we write the value "0" at the beginning of the counting process.

Determining the number of functions for each category of each level of complexity is based on the following rules (numerical constants for constructing these rules are recommended in [5]):

- for input functions ($i = \overline{1..qt_{fi}}$):

1. $qt_{fi_s} = qt_{fi_s} + 1 \quad \forall (qted_{fi_i} \in \{1,...,19\}) \wedge (qtf_{fi_i} \in \{0,1\})$
2. $qt_{fi_s} = qt_{fi_s} + 1 \quad \forall (qted_{fi_i} \in \{1,...,5\}) \wedge (qtf_{fi_i} \in \{2,3\})$
3. $qt_{fi_m} = qt_{fi_m} + 1 \quad \forall (qted_{fi_i} \in \{1,...,5\}) \wedge (qtf_{fi_i} \geq 4)$
4. $qt_{fi_m} = qt_{fi_m} + 1 \quad \forall (qted_{fi_i} \in \{6,...,19\}) \wedge (qtf_{fi_i} \in \{2,3\})$
5. $qt_{fi_m} = qt_{fi_m} + 1 \quad \forall (qted_{fi_i} \geq 20) \wedge (qtf_{fi_i} \in \{0,1\})$
6. $qt_{fi_c} = qt_{fi_c} + 1 \quad \forall (qted_{fi_i} \geq 20) \wedge (qtf_{fi_i} \in \{2,3\})$
7. $qt_{fi_c} = qt_{fi_c} + 1 \quad \forall (qted_{fi_i} \geq 6) \wedge (qtf_{fi_i} \geq 4)$

- for output functions ($j = \overline{1..qt_{fo}}$):

8. $qt_{fo_s} = qt_{fo_s} + 1 \quad \forall (qted_{fo_j} \in \{1,...,19\}) \wedge (qtf_{fo_j} \in \{0,1\})$
9. $qt_{fo_s} = qt_{fo_s} + 1 \quad \forall (qted_{fo_j} \in \{1,...,5\}) \wedge (qtf_{fo_j} \in \{2,3\})$
10. $qt_{fo_m} = qt_{fo_m} + 1 \quad \forall (qted_{fo_j} \in \{1,...,5\}) \wedge (qtf_{fo_j} \geq 4)$
11. $qt_{fo_m} = qt_{fo_m} + 1 \quad \forall (qted_{fo_j} \in \{6,...,19\}) \wedge (qtf_{fo_j} \in \{2,3\})$

12. $qt_{fo_m} = qt_{fo_m} + 1 \quad \forall (qted_{fo_j} \geq 20) \wedge (qtf_{fo_j} \in \{0,1\})$
13. $qt_{fo_c} = qt_{fo_c} + 1 \quad \forall (qted_{fo_j} \geq 20) \wedge (qtf_{fo_j} \in \{2,3\})$
14. $qt_{fo_c} = qt_{fo_c} + 1 \quad \forall (qted_{fo_j} \geq 6) \wedge (qtf_{fo_j} \geq 4)$

- for input requests ($k = \overline{1..qt_{cfi}}$):

15. $qt_{cfi_s} = qt_{cfi_s} + 1 \quad \forall (qted_{cfi_k} \in \{1,...,19\}) \wedge (qtf_{cfi_k} \in \{0,1\})$
16. $qt_{cfi_s} = qt_{cfi_s} + 1 \quad \forall (qted_{cfi_k} \in \{1,...,5\}) \wedge (qtf_{cfi_k} \in \{2,3\})$
17. $qt_{cfi_m} = qt_{cfi_m} + 1 \quad \forall (qted_{cfi_k} \in \{1,...,5\}) \wedge (qtf_{cfi_k} \geq 4)$
18. $qt_{cfi_m} = qt_{cfi_m} + 1 \quad \forall (qted_{cfi_k} \in \{6,...,19\}) \wedge (qtf_{cfi_k} \in \{2,3\})$
19. $qt_{cfi_m} = qt_{cfi_m} + 1 \quad \forall (qted_{cfi_k} \geq 20) \wedge (qtf_{cfi_k} \in \{0,1\})$
20. $qt_{cfi_c} = qt_{cfi_c} + 1 \quad \forall (qted_{cfi_k} \geq 20) \wedge (qtf_{cfi_k} \in \{2,3\})$
21. $qt_{cfi_c} = qt_{cfi_c} + 1 \quad \forall (qted_{cfi_k} \geq 6) \wedge (qtf_{cfi_k} \geq 4)$

- for output requests ($g = \overline{1..qt_{cfo}}$):

22. $qt_{cfo_s} = qt_{cfo_s} + 1 \quad \forall (qted_{cfo_g} \in \{1,...,19\}) \wedge (qtf_{cfo_g} \in \{0,1\})$
23. $qt_{cfo_s} = qt_{cfo_s} + 1 \quad \forall (qted_{cfo_g} \in \{1,...,5\}) \wedge (qtf_{cfo_g} \in \{2,3\})$
24. $qt_{cfo_m} = qt_{cfo_m} + 1 \quad \forall (qted_{cfo_g} \in \{1,...,5\}) \wedge (qtf_{cfo_g} \geq 4)$
25. $qt_{cfo_m} = qt_{cfo_m} + 1 \quad \forall (qted_{cfo_g} \in \{6,...,19\}) \wedge (qtf_{cfo_g} \in \{2,3\})$
26. $qt_{cfo_m} = qt_{cfo_m} + 1 \quad \forall (qted_{cfo_g} \geq 20) \wedge (qtf_{cfo_g} \in \{0,1\})$
27. $qt_{cfo_c} = qt_{cfo_c} + 1 \quad \forall (qted_{cfo_g} \geq 20) \wedge (qtf_{cfo_g} \in \{2,3\})$
28. $qt_{cfo_c} = qt_{cfo_c} + 1 \quad \forall (qted_{cfo_g} \geq 6) \wedge (qtf_{cfo_g} \geq 4)$

- for files ($n = \overline{1..qt_f}$):

29. $qt_{f_s} = qt_{f_s} + 1 \quad \forall (qted_{f_n} \in \{1,...,50\}) \wedge (qtlr_{f_n} = 1)$
30. $qt_{f_s} = qt_{f_s} + 1 \quad \forall (qted_{f_n} \in \{1,...,19\}) \wedge (qtlr_{f_n} \in \{2,...,5\})$
31. $qt_{f_m} = qt_{f_m} + 1 \quad \forall (qted_{f_n} \in \{1,...,19\}) \wedge (qtlr_{f_n} \geq 6)$
32. $qt_{f_m} = qt_{f_m} + 1 \quad \forall (qted_{f_n} \in \{20,...,50\}) \wedge (qtlr_{f_n} \in \{2,...,5\})$
33. $qt_{f_m} = qt_{f_m} + 1 \quad \forall (qted_{f_n} \geq 51) \wedge (qtlr_{f_n} = 1)$
34. $qt_{f_c} = qt_{f_c} + 1 \quad \forall (qted_{f_n} \geq 51) \wedge (qtlr_{f_n} \in \{2,...,5\})$
35. $qt_{f_c} = qt_{f_c} + 1 \quad \forall (qted_{f_n} \geq 20) \wedge (qtlr_{f_n} \geq 6)$

- for interfaces ($l = \overline{1..qt_i}$):

36. $qt_{i_s} = qt_{i_s} + 1 \quad \forall (qted_{i_l} \in \{1,...,50\}) \wedge (qtlr_{i_l} = 1)$
37. $qt_{i_s} = qt_{i_s} + 1 \quad \forall (qted_{i_l} \in \{1,...,19\}) \wedge (qtlr_{i_l} \in \{2,...,5\})$
38. $qt_{i_m} = qt_{i_m} + 1 \quad \forall (qted_{i_l} \in \{1,...,19\}) \wedge (qtlr_{i_l} \geq 6)$
39. $qt_{i_m} = qt_{i_m} + 1 \quad \forall (qted_{i_l} \in \{20,...,50\}) \wedge (qtlr_{i_l} \in \{2,...,5\})$
40. $qt_{i_m} = qt_{i_m} + 1 \quad \forall (qted_{i_l} \geq 51) \wedge (qtlr_{i_l} = 1)$
41. $qt_{i_c} = qt_{i_c} + 1 \quad \forall (qted_{i_l} \geq 51) \wedge (qtlr_{i_l} \in \{2,...,5\})$
42. $qt_{i_c} = qt_{i_c} + 1 \quad \forall (qted_{i_l} \geq 20) \wedge (qtlr_{i_l} \geq 6)$

6) check the number of functions in each category according to the following rules:

- if $qt_{fi} \neq qt_{fi_s} + qt_{fi_m} + qt_{fi_c}$, then an error occurred when calculating the number of input functions by difficulty levels ($a = a + 1$);
- if $qt_{fo} \neq qt_{fo_s} + qt_{fo_m} + qt_{fo_c}$, then an error occurred when calculating the number of output functions by difficulty levels ($a = a + 1$);

- if $qt_{cfi} \neq qt_{cfi_s} + qt_{cfi_m} + qt_{cfi_c}$, then an error occurred when calculating the number of input requests by difficulty levels ($a = a + 1$);
- if $qt_{cfo} \neq qt_{cfo_s} + qt_{cfo_m} + qt_{cfo_c}$, then an error occurred when calculating the number of output requests by difficulty levels ($a = a + 1$);
- if $qt_f \neq qt_{f_s} + qt_{f_m} + qt_{f_c}$, then an error occurred when counting the number of files by difficulty levels ($a = a + 1$);
- if $qt_i \neq qt_{i_s} + qt_{i_m} + qt_{i_c}$, then an error occurred when calculating the number of interfaces by difficulty levels ($a = a + 1$);

7) if $a \neq 0$, then go back to step 5, else, if $a = 0$, go to step 8;

8) determination of the number of functional points for the functions of each category according to the following formulas (using the weights of complexity given in [5]):

- for input functions: $qt_{fi_{FP}} = 3 \cdot qt_{fi_s} + 4 \cdot qt_{fi_m} + 6 \cdot qt_{fi_c}$;
- for output functions: $qt_{fo_{FP}} = 4 \cdot qt_{fo_s} + 5 \cdot qt_{fo_m} + 7 \cdot qt_{fo_c}$;
- for input requests: $qt_{cfi_{FP}} = 3 \cdot qt_{cfi_s} + 4 \cdot qt_{cfi_m} + 6 \cdot qt_{cfi_c}$;
- for output requests: $qt_{cfo_{FP}} = 4 \cdot qt_{cfo_s} + 5 \cdot qt_{cfo_m} + 7 \cdot qt_{cfo_c}$;
- for files: $qt_{f_{FP}} = 7 \cdot qt_{f_s} + 10 \cdot qt_{f_m} + 15 \cdot qt_{f_c}$;
- for interfaces: $qt_{i_{FP}} = 5 \cdot qt_{i_s} + 7 \cdot qt_{i_m} + 10 \cdot qt_{i_c}$;

9) determination of the total number of functional points:

$$qt_{FP} = qt_{fi_{FP}} + qt_{fo_{FP}} + qt_{cfi_{FP}} + qt_{cfo_{FP}} + qt_{f_{FP}} + qt_{i_{FP}}$$

10) determining the user's need to correct the total number of functional points taking into account environmental factors that affect the software development process as a whole: if such adjustment is necessary, then steps 11-14 are performed, otherwise there is a transition to step 15;

11) assessment of environmental factors (detailed description of factors is given in [5]): fe_{cdt} - assessment of data transmission channels; fe_{dc} - evaluation of distributed computing; $fe_{p\backslash}$ - assessment of performance requirements; fe_c - evaluation of configuration with restrictions; fe_t - estimation of transaction frequency; fe_{ir} - evaluation of interactive request; fe_u - evaluation of efficiency at the end user level; fe_{iu} - evaluation of interactive updates; fe_{cp} - assessment of complex processing; fe_{ru} - assessment of reuse; fe_i - evaluation of conversion / installation simplification; fe_o - assessment of operation simplification; fe_{un} - assessment of use at several nodes; fe_{pf} - assessment of the potential to change the function. Evaluation is on a scale from 0 to 5, where a value of 0 means the impossibility of applying the factor [5;

12) determination of the general assessment of environmental factors:

$$fe = fe_{cdt} + fe_{dc} + fe_p + fe_c + fe_t + fe_{ir} + fe_u + fe_{iu} + fe_{cp} + fe_{ru} + fe_i + fe_o + fe_{un} + fe_{pf}$$

13) determining the correction factor of complexity [5]:

$$cfc = 0{,}65 + (0{,}01 \cdot fe)$$

14) correcting the number of function points:

$$qt_{FP\,correct} = qt_{FP} \cdot cfc$$

15) projected estimate of the LOC indicator for this software project with its potential implementation in different programming languages (average LOC indicators for different programming languages per one function point are given in [5]):

$$LOC_l = qt_{FP\,correct} \cdot LOC_{l\,av},$$

where $LOC_{l\,av}$ - the average LOC for the programming language l, taken from [5]; LOC_l - evaluation of the LOC indicator for this software project in language l; $qt_{FP\,correct}$ is replaced by an indicator qt_{FP}, if the correction was not performed.

The developed method of estimating the laboriousness on the basis of functional points differs from the existing methods in that it is formalized, due to which it eliminates the dependence of evaluation on the subjects involved in the evaluation process. Advantages of the developed method: the possibility of its use in the early stages of the software life cycle; no dependence on the used programming language, methodology and technology; the ability to measure LOC-estimates for project implementation in different programming languages; the possibility of applying this method for software systems with a graphical user interface; taking into account environmental factors as needed.

Let's consider an example of the application of the developed method of estimating the laboriousness based on functional points. Let's consider the specification of requirements for the software project of the self-organized distributed system of detection of the malicious software in computer networks.

For this project, the number of functions by category are:

$qt_{fi} = 100$; $qt_{fo} = 105$; $qt_{cfi} = 50$; $qt_{cfo} = 62$; $qt_{f} = 90$; $qt_{i} = 89$;

the numbers of data elements by category are:

$QTED_{fi} = \{1,3,5,7,9,21,25,26,34,3,6,8,9,10,11,12,18,21,22,1,3,5,7,9,2,4,6,8,11,13,1,5,1,7,19,15,17,18,16,14,12,13,2,23,$
$24,25,3,35,36,30,3,6,8,9,10,11,12,18,21,1,3,5,7,9,21,25,26,34,5,7,9,2,4,6,8,11,13,1,5,1,7,19,15,17,18,16,14,12,13,2,23,$;
$24,25,3,35,36,30\}$

$QTED_{fo} = \{24,25,3,35,36,30,3,6,8,9,10,11,12,18,21,1,3,5,7,9,21,25,26,34,5,7,9,2,4,6,8,11,13,1,5,1,7,19,15,17,18,16,$
$14,12,13,2,23,1,3,5,7,9,21,25,26,34,3,6,8,9,10,11,12,18,21,22,1,3,5,7,9,2,4,6,8,11,13,1,5,1,7,19,15,17,18,16,14,12,13,2,$;
$23,24,25,3,35,36,30,24,25,3,35,36\}$

$QTED_{cfi} = \{23,24,25,3,35,36,30,24,25,3,35,36,24,25,3,35,36,30,3,6,8,9,10,11,12,18,21,1,3,5,7,9,21,25,26,34,5,7,$;
$9,2,4,6,8,11,13,1,5,1,7,19\}$

$QTED_{cfo} = \{21,1,3,5,7,9,21,25,26,34,5,7,9,2,4,6,8,11,13,1,5,1,7,19,23,24,25,3,35,36,30,24,25,3,35,36,24,25,3,35,$;
$36,30,3,6,8,9,10,11,12,18,23,24,25,3,35,36,30,24,25,3,35,36\}$

$QTED_{f} = \{6,8,9,10,11,12,18,21,22,1,3,5,7,9,2,4,6,8,11,13,1,5,1,7,19,15,17,18,16,14,12,13,2,23,24,25,3,35,36,30,3,$;
$6,8,9,10,11,12,18,21,1,3,5,7,9,21,25,26,34,5,7,9,2,4,6,8,11,13,1,5,1,7,19,15,17,18,16,14,12,13,2,23,24,25,3,35,36,30\}$

$QTED_{i} = \{23,24,25,3,35,36,30,24,25,3,35,36,24,25,3,35,36,30,3,6,8,9,10,11,12,18,21,1,3,5,7,9,21,25,26,34,5,7,$
$9,2,4,6,8,11,13,1,5,1,7,19,2,23,24,25,3,35,36,30,24,25,3,35,36,2,23,24,25,3,35,36,30,24,25,3,35,36,2,23,24,25,$;
$3,35,36,30,24,25,3,35,36\}$

the numbers of involved files by category are:

$QTF_{fi} = \{2,3,1,4,5,1,2,3,4,5,4,5,3,2,1,3,2,4,6,1,4,5,3,2,1,3,2,4,6,1,2,3,1,4,5,1,2,3,4,5,2,3,1,4,5,1,2,3,4,5,4,5,3,2,1,3,$;
$2,4,6,1,4,5,3,2,1,3,2,4,6,1,2,3,1,4,5,1,2,3,4,5,2,3,1,4,5,1,2,3,4,5,4,5,3,2,1,3,2,4,6,1\}$

$QTF_{fo} = \{4,3,5,6,1,2,3,4,5,3,5,4,4,4,2,1,3,2,3,4,5,3,2,1,4,4,3,5,6,1,2,3,4,5,3,5,4,4,4,2,1,3,2,3,4,5,3,2,1,4,$;
$4,3,5,6,1,2,3,4,5,3,5,4,4,4,2,1,3,2,3,4,5,3,2,1,4,4,3,5,6,1,2,3,4,5,3,5,4,4,4,2,1,3,2,3,4,5,3,2,1,4,3,2,1,4,4\}$

$QTF_{cfi} = \{0,2,3,1,4,5,2,1,2,2,1,1,0,2,3,1,4,5,2,1,2,2,1,1,0,2,3,1,4,5,2,1,2,2,1,1,0,2,3,1,4,5,2,1,2,2,1,1,3,2\}$;

$QTF_{cfo} = \{1,2,1,0,0,3,4,2,1,3,2,1,6,5,1,2,1,0,0,3,4,2,1,3,2,1,6,5,1,2,1,0,0,3,4,2,1,3,2,1,6,5,1,2,1,0,0,3,4,2,1,3,2,1,6,5,3,4,5,6,1,2\}$;

the numbers of logical records of the "format-relationship" type by category are:

$QTLR_{fl} = \{1,2,5,4,6,7,4,3,2,1,2,4,1,8,1,2,5,4,6,7,4,3,2,1,2,4,1,8,1,2,5,4,6,7,4,3,2,1,2,4,1,8,1,2,$;
$5,4,6,7,4,3,2,1,2,4,1,8,1,2,5,4,6,7,4,3,2,1,2,4,1,8,1,2,3,4,5,6\}$

$QTLR_{i} = \{1,6,7,5,4,7,8,4,3,4,5,6,2,1,5,1,6,7,5,4,7,8,4,3,4,5,6,2,1,5,1,6,7,5,4,7,8,4,3,4,5,6,$
$2,1,5,1,6,7,5,4,7,8,4,3,4,5,6,2,1,5,1,6,7,5,4,7,8,4,3,4,5,6,2,1\}$.

Let's determine the number of functions of each category of each level of complexity based on the rules of step 5.

For input functions, the following numbers of functions are available by difficulty levels:

$qt_{fi_s} = 25$; $qt_{fi_m} = 17$; $qt_{fi_c} = 58$.

For output functions:

$qt_{fo_s} = 26$; $qt_{fo_m} = 18$; $qt_{fo_c} = 61$.

For input requests:

$qt_{cfi_s} = 13$; $qt_{cfi_m} = 8$; $qt_{cfi_c} = 29$.

For output requests:

$qt_{cfo_s} = 16$; $qt_{cfo_m} = 10$; $qt_{cfo_c} = 36$.

For files:

$qt_{f_s} = 23$; $qt_{f_m} = 15$; $qt_{f_c} = 52$.

For interfaces:

$qt_{i_s} = 22$; $qt_{i_m} = 15$; $qt_{i_c} = 52$.

Checking the number of functions in each category showed that there are no errors in calculating the number of functions of different categories by difficulty levels, then let's go to step 8 and let's determine the number of function points for the functions of each category:

- for input functions:

$qt_{fi\,FP} = 3 \cdot 25 + 4 \cdot 17 + 6 \cdot 58 = 491;$

- for output functions:

$qt_{fo\,FP} = 4 \cdot 26 + 5 \cdot 18 + 7 \cdot 61 = 621;$

- for input requests:

$qt_{cfi\,FP} = 3 \cdot 13 + 4 \cdot 8 + 6 \cdot 29 = 245;$

- for output requests:

$qt_{cfo\,FP} = 4 \cdot 16 + 5 \cdot 10 + 7 \cdot 36 = 366;$

- for files:

$qt_{f\,FP} = 7 \cdot 23 + 10 \cdot 15 + 15 \cdot 52 = 1091;$

- for interfaces:

$qt_{i\,FP} = 5 \cdot 22 + 7 \cdot 15 + 10 \cdot 52 = 735.$

Then the total number of function points is:

$qt_{FP} = 491 + 621 + 245 + 366 + 1091 + 735 = 3549.$

The results of the experiment showed that the considered project has the complexity of development at 3549 functional points. For this project there is no need to adjust the total number of functional points taking into account environmental factors, so we can estimate the LOC for this software project in its implementation in different programming languages.

Conclusions

Assessing the laboriousness of the software development process of computer systems is one of the most important activities in the software development process. In the absence of an adequate and reliable assessment, it is impossible to ensure clear project planning and management. Underestimation of the cost, duration and resources required to create the software entails an insufficient number of project team, excessively short development time and, as a result, loss of confidence in the developers in case of schedule violations.

The method for estimating the laboriousness of software development based on functional points is proposed, which makes it possible to determine the number of functional points for a software project, and also allows to estimate the size of a software project (LOC-assessment) at its early stages of its life cycle.

The presented method provides the process of estimating the laboriousness of software development based on functional points on a mathematical basis, thus eliminating the dependence of evaluation on the entities involved in the evaluation process, and solves the problem of single use of estimating the laboriousness of software development based on functional points.

The developed method of estimating the laboriousness on the basis of functional points differs from the existing methods in that it is formalized. Advantages of the developed method: the possibility of its use in the early stages of the software life cycle; no dependence on the used programming language, methodology and technology; the ability to measure LOC-estimates for project implementation in different programming languages; the possibility of applying this method for software systems with a graphical user interface; taking into account environmental factors as needed.

The results of the experiment showed that the considered project of a self-organized distributed system for detecting malicious software in computer networks has the complexity of developing 3549 functional points.

Prospects for further research are:

1) development of a method (algorithm) for estimating environmental factors;

2) development of a method for estimating the laboriousness of software development based on property points;

3) development of a method (algorithm) for estimating the duration and cost of software development on the basis of LOC-estimates obtained by the developed methods;

4) development of a decision support system for estimating the laboriousness of software development based on functional points and property points, which will be based on the developed mathematical methods.

References

1. Latest study shows rise in project failures. Web-site. URL: http://kinzz.com/resources/articles/91-project-failures-rise-study-shows (Last accessed: October 7, 2021).
2. The Standish Group International: CHAOS Manifesto – Think big, act small. Technical report, CHAOS Knowledge Center (2013). Web-site. URL: http://www.versionone.com/assets/img/files/CHAOSManifesto2013.pdf (Last accessed: October 7, 2021).
3. Hastie Shane, Wojewoda Stéphane. Standish Group 2015 Chaos Report – Q&A with Jennifer Lynch. Web-site. URL: http://www.infoq.com/articles/standish-chaos-2015 (Last accessed: October 7, 2021).
4. The Standish Group Report: CHAOS 2014. Web-site. URL: https://www.projectsmart.co.uk/white-papers/chaos-report.pdf (Last accessed: October 7, 2021).
5. R. T. Futrell, D. F. Shafer, L. Shafer. Quality Software Project Management. Prentice Hall Professional, 2020. 1639 p.